Tarea 2

Tries para autocompletado

Profesores: Benjamín Bustos y Gonzalo Navarro
Auxiliares: Diego Arias y Claudio Gaete

1. Indicaciones administrativas

- Fecha de entrega: jueves 6 de noviembre a las 23:59.
- Grupos de a lo más 3 personas de la misma sección.
- Lenguaje a utilizar: C, C++ o Java.

2. Contexto

El objetivo de esta tarea es implementar el trie visto en clases y extenderlo con funcionalidades que permitan utilizarlo como motor de autocompletado de texto.

Se insertarán palabras provenientes de un dataset, luego se simulará la escritura de texto descendiendo por el trie para cada palabra. Se implementarán dos variantes con criterios distintos: una que retorna la palabra del subárbol que fue accedida de forma más **reciente**, y otra que retorna la palabra con mayor **frecuencia** de accesos.

3. Trie

Como recordatorio, un trie es un árbol en el que cada arista está etiquetada con un carácter y cada nodo representa un prefijo de alguna palabra del conjunto. La raíz corresponde al prefijo vacío, y recorrer desde la raíz hasta un nodo concatena las etiquetas y produce el prefijo asociado. Un nodo puede marcar el fin de palabra (indicando que el prefijo es una palabra válida) y en este caso agregaremos metadatos para realizar un autocompletar.

Para la funcionalidad del trie y habilitar autocompletado de las dos variantes, cada nodo debe mantener la siguiente información actualizada:

1. `parent`: una referencia a su nodo padre (nulo si el nodo es la raíz).
2. `next`: estructura que mapea caracteres Σ a hijos.
3. `priority`: dependiendo del criterio a utilizar, es el tiempo de acceso o la cantidad de accesos a este nodo terminal[1].
4. `str`: si el nodo es terminal esto debe contener un puntero al string asociado; no es estrictamente necesario en implementaciones de trie, pero facilita la experimentación.
5. `best_terminal`: un puntero al nodo terminal del subárbol con mayor prioridad.
6. `best_priority`: la prioridad del nodo con mayor prioridad del subárbol.

[1]Denominamos un nodo terminal como aquel accedido mediante el carácter especial \$, que indica el final de la palabra.

Como se utilizarán datasets del inglés, se tiene que $\Sigma = 27$ (26 letras y el carácter especial \$, que indica el final de la palabra), por lo que `next` será un arreglo de tamaño fijo Σ, con punteros al siguiente nodo o nulo en caso de no existir.

Para esta tarea consideraremos las siguientes operaciones:

1. `insert(w)`: inserta la palabra w carácter por carácter, creando nodos cuando sea necesario.
2. `descend(v, c)`: dado un puntero a un nodo v en el trie, retorna un puntero al nodo asociado a descender por el carácter c o nulo en caso de no existir.
3. `autocomplete(v)`: retorna un puntero al nodo terminal que representa el mejor autocompletado según el subárbol de v.
4. `update_priority(v)`: actualiza la prioridad del nodo terminal v según la variante y actualiza los nodos en el camino a la raíz.
   - En el caso de la variante de frecuencia deben sumarle a uno al `priority`, indicando que fue utilizada una vez más.
   - Caso contrario en la variante de reciente, será necesario asignarle a `priority` algún timestamp. Pueden tener un contador a nivel trie que marque la cantidad de accesos ocurridos hasta ahora, pueden sumarle uno y lo asignan a `priority`.
   - Si la `best_priority` del nodo padre es menor a la prioridad actualizada del hijo, entonces se debe actualizar `best_priority` y `best_terminal` para el padre. Esta actualización se puede repetir varias veces hasta llegar a la raíz, o hasta que se alcance un nodo cuya `best_priority` sea mayor a la prioridad actualizada del hijo.

Es importante mencionar que no se utilizará compresión tipo Patricia, por lo tanto, cada inserción avanza carácter a carácter, creando nodos en caso de ser necesario.

Adicionalmente, para la experimentación será necesario conocer el consumo de memoria de su trie. Para esto, guarden una variable **a nivel trie** que mantenga el número de nodos que está utilizando la estructura. Para mantenerlo actualizado, deben aumentar el contador en 1 cada vez que se crea un nodo en `insert(w)`.

4. Experimentación

Se entrega un dataset de palabras en el archivo `words.txt`, que contiene $N = 2^{18}$ palabras. Para la experimentación, deberán insertar todas las palabras en su trie y luego se simulará el proceso que haría un usuario para escribir.

4.1. Consumo de memoria

A medida que insertan el dataset en el trie, deberán medir el consumo de memoria; en particular, en la inserción $i \in \{2^0, 2^1, ..., 2^{17}, N\}$ deberán medir el consumo de memoria de su estructura. Esto se medira en base a la cantidad de nodos que tiene su trie, el cual tienen que obtener a partir lo especificado al final de la Sección 3.

Realicen un gráfico con el número de nodos, normalizado según la cantidad de **caracteres** insertados hasta ese momento.

4.2. Tiempo

De forma similar, para la medición de tiempo deben medir cuánto tarda en insertarse cada grupo de $\frac{N}{M}$ palabras con $M = 16$ (es decir, cuánto tardan en insertarse las primeras $\frac{N}{M}$ palabras, luego cuánto tardan las siguientes $\frac{N}{M}$, y así), y luego normalicen estos tiempos según la cantidad de caracteres de cada grupo.

El gráfico resultante debe mostrar cómo evoluciona el tiempo de inserción por carácter a medida que crece el trie.

4.3. Análisis de autocompletado

Finalmente, se medirá la capacidad de autocompletado de su estructura con ambas variantes. Para ello, se tomarán varios textos de referencia y se simulará su escritura palabra por palabra, letra por letra.

Se busca medir cuántas letras se habría ahorrado un usuario que utilizara este esquema para escribir texto, asumiendo que en el momento en que el usuario vea su palabra entonces no escribe más letras. Dada una palabra w se realiza lo siguiente:

- Se desciende el trie utilizando `descend(v, c)` para cada carácter del string w.
  - Si retorna nulo, entonces la palabra w no pertenece al trie y el usuario habría tenido que escribirla completa, por lo que se suma $|w|$ a la cantidad de caracteres que tendría que haber escrito el usuario.
  - Si no retorna nulo, entonces se llama `autocomplete` sobre el puntero retornado, aquí hay dos casos:
    - El `str` guardado en el nodo terminal es igual a w (más el caracter \$), en este caso el autocompletado fue exitoso y la cantidad de caracteres que el usuario tuvo que escribir fue la cantidad de llamadas a `descend`. Se pasa a la siguiente palabra.
    - De lo contrario, necesitamos más input y se vuelve a llamar `descend` con el nodo actual y el siguiente carácter. Si ya ocupamos el último carácter, el autocompletado no funcionó y se debe sumar $|w|$.
- Antes de pasar a la siguiente palabra deben utilizar `update_priority` para actualizar la prioridad de w según la variante (en caso de que w no pertenezca no se actualiza).

Deberán simular este proceso sobre varios datasets para comparar de $L = 2^{22}$ palabras: `wikipedia.txt`, que contiene texto real de distintas páginas de Wikipedia, `random.txt`, que usa palabras del mismo dataset de `words.txt` con distribución uniforme, y `random_with_distribution.txt`, que también es generada de forma aleatoria pero con una distribución más realista.

Similarmente a la Sección 4.1, para la palabra i-ésima escrita por el usuario con $i \in \{2^0, 2^1, ..., 2^{21}, L\}$, calculen el porcentaje de caracteres escritos por el usuario respecto al total del texto (ignorando espacios) y grafiquen estos valores para cada variante en cada dataset.

Finalmente, deberán comparar los tiempos de ejecución de su estructura realizando el proceso de autocompletado con cada variante y cada dataset. Vean tiempos totales, tiempos normalizados por la cantidad de palabras y también normalizados según el número de caracteres.

5. (Bonus) Interfaz interactiva

Finalmente, estará disponible como puntaje adicional la posibilidad de escribir una interfaz interactiva que le permita utilizar su propio trie para redactar. Debe contemplar la siguiente funcionalidad:

- Escoger al iniciar la variante del trie a ser utilizada: reciente o frecuencia.
- Se inicializa con el mismo dataset de la experimentación.
- Al escribir un prefijo debe mostrar la palabra con mayor prioridad del subarbol asociado.
- Debe existir una forma de aceptar la "recomendación", como por ejemplo `TAB` que aplique la palabra y permita comenzar a escribir la siguiente.
- Adicionalmente, debe estar la opción de ignorar las recomendaciones y escribir el prefijo actual con por ejemplo `ENTER`.
- Por último, también debe existir una forma de borrar caracteres del prefijo y que esto no rompa la correctitud (que se suba en el trie).

Puede ser implementado como una aplicación con interfaz por la línea de comandos, o también a nivel GUI en caso de preferirlo.

Esta bonificación será sumada a la **nota de código** de la tarea; si esta ya es un 7 y aún se tiene bonificación, se podrá sumar el restante a la **nota de informe**.

6. Entregables

Se deberá entregar el código y un informe donde se explique el experimento en estudio. Con esto se obtendrá una nota de código ($NCod$) y una nota de informe ($NInf$). La nota final de la tarea será el promedio simple entre ambas notas ($NT_1 = 0.5 \cdot NCod + 0.5 \cdot NInf$).

6.1. Código

La entrega de código tiene que contener:

- **(0.3 pts)** README: Archivo con las instrucciones para ejecutar el código, debe ser lo suficientemente explicativo para que cualquier persona solo leyendo el README pueda ejecutar la totalidad de su código (incluyendo librerías no entregadas por el equipo docente que potencialmente se deban instalar).
- **(0.2 pts)** Firmas: Cada estructura de datos y función debe tener una descripción de lo que hace y una descripción de sus parámetros de entrada y salida.

- **(1.5 pts)** Implementación del trie: La inserción y búsqueda en el trie es correcta.
- **(1.5 pts)** Implementación de prioridad: El cálculo de la prioridad y el autocompletado es correcto para cada nodo.
- **(0.5 pts)** Experimento: Se realiza la experimentación utilizando los datasets pedidos.
- **(1.5 pts)** Obtención de resultados: La forma en el que se obtienen los resultados es correcta. Esta sección se divide en:
  - **(0.5 pts)** Medición de memoria (cantidad de nodos).
  - **(0.4 pts)** Medición de tiempos de ejecución.
  - **(0.6 pts)** Análisis de autocompletado.
- **(0.5 pts)** Main: Un archivo o parte del código (función main) que permita ejecutar la construcción y "escritura".
- **Bonus (1.5 pts)** Interfaz interactiva: Se implementa la interfaz según lo indicado en la Sección 5.

6.2. Informe

El informe debe ser claro y conciso. Se recomienda hacerlo en LaTeX o Typst. Debe contener:

- **(0.8 pts)** Introducción: Presentar el tema en estudio, resumir lo que se dirá en el informe y presentar una hipótesis.
- **(0.8 pts)** Desarrollo: Presentación de algoritmos, estructuras de datos, cómo funcionan y por qué. Recordar que los métodos ya son conocidos por el equipo docente, lo que importa son sus propias implementaciones (qué decisiones tomaron que no están mencionadas en el enunciado).
- **(2.4 pts)** Resultados: Especificación de los datos que se utilizaron para los experimentos, la cantidad de veces que se realizaron los tests, con qué inputs, qué tamaño, etc. Se debe mencionar el sistema operativo y los tamaños de sus cachés y RAM con los que se ejecutaron los experimentos. Se deben mostrar gráficos/tablas con la información solicitada en la Sección 4 y mencionar solo lo que se puede observar de estos.
- **(1.2 pts)** Análisis: Comentar y concluir sus resultados. Se hacen las inferencias de sus resultados, aplicando conocimientos del curso.
- **(0.8 pts)** Conclusión: Recapitulación de lo que se hizo, se concluye lo que se puede decir con respecto a sus resultados. También ven si su hipótesis se cumplió o no y analizan la razón. Por último, se menciona qué se podría mejorar en su desarrollo en una versión futura, qué falta en su documento, qué no se ha resuelto y cómo se podrían extender.

7. Informaciones útiles

- Les recomendamos implementar la función `update_priority(v)` de forma genérica (si es que su lenguaje les permite) o que la opción este detras de una variable. Así pueden utilizar la misma estructura, pero parametrizada según la variante.
- Antes de que ejecuten la experimentación, recomendamos que prueben unos casos simples primeros para validar que su trie retorna las recomendaciones correctas en ambas variantes.